실험 I. 간단한 입출력 실험-1

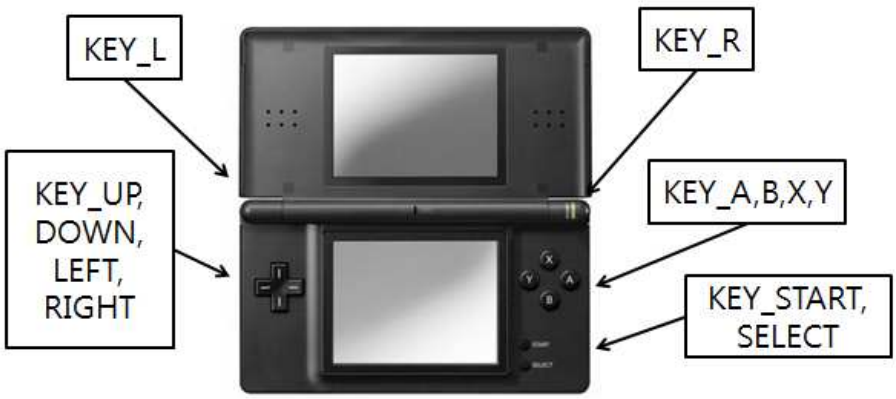
1. 목적

(1) 기본적인 RTOS 기반 프로그램 수행 방식을 따라해 본다.
(2) 입출력 장치를 Access 하기 위한 API를 따라해 본다.
(2) 스위치를 읽어 닌텐도 화면 상의 BAR LED를 다양한 방법으로 켜는 방법을 배운다.

2. 실험 배경

컴퓨터 시스템에서 모든 입출력 장치는 기본적으로 메모리 또는 입출력 번지를 CPU가 참조함으로써 Access가 가능하다. 우리 실험에 사용하는 닌텐도 DS 역시 같은 방식으로 스위치, LCD 등 장치에 접근할 수 있다. 하지만 이번 실험은 프로그램 개발 환경에 익숙해지기 위하여 다음과 같이 이미 만들어진 API를 통해 입출력 장치에 접근하는 방식으로 프로그램을 작성해 봄으로써, 앞으로의 실험 환경에 익숙해지고자 한다.

2.1 닌텐도의 각종 스위치를 읽기 위한 API



헤더 파일 ./source/realio.h에는 닌텐도 하드웨어가 지원하는 12개의 스위치 상태를 읽기 위한 API가 정의되어 있다. 실제 스위치 값은 keyCurrent() 라는 NDS library 함수를 통해 하드웨어를 읽게 되며. 이 함수의 리터값 아래쪽 16비트의 각 비트에 12개의 스위치가 매핑되어 있

다. 스위치가 눌리면 1, 눌리지 않았으면 0이다.

따라서 NDS_SWITCH() 매크로는 모든 스위치의 상태를 리턴하며, 각 스위치 별로 한 비트씩 할당되어 있다. KEY_UP, KEY_DOWN, KEY_LEFT, KEY_RIGHT는 아래쪽 LCD 왼쪽의 방향키이며 KEY_L, KEY_R은 뒤편의 왼쪽, 오른쪽 버튼이다.

```
#define KEY_A           0x0001
#define KEY_B           0x0002
#define KEY_X           0x0400
#define KEY_Y           0x0800
#define KEY_L           0x0200
#define KEY_R           0x0100
#define KEY_START       0x0008
#define KEY_SELECT      0x0004
#define KEY_UP          0x0040
#define KEY_DOWN        0x0080
#define KEY_LEFT        0x0020
#define KEY_RIGHT       0x0010

#define NDS_SWITCH()   (keysCurrent() & 0xFFFF)
```

프로그램에서는 다음과 같이 NDS_SWITCH()와 KEY_??? 매크로를 논리 AND (&) 하여 어떤 키가 눌렸는지 확인할 수 있다.

```
if (NDS_SWITCH() & KEY_START) {
    ...
}
```

2.2 가상 디바이스인 Bar LED를 켜기 위한 API

닌텐도 DS에는 두 개의 LED가 있는데, 이 정도로는 실험에 부족하여 가상적으로 두 개의 Bar LED를 구성하였다.

이 Bar LED는 각각 8개씩의 LED로 구성되어 있으며, 아래 박스와 같이 가상 주소 0x408, 0x40C 번지에 각각 BARLED1, BARLED2 이름으로 존재한다. 이 주소는 가상 주소이지만, writeb_virtual_io(u32 addr, u8 value) 함수를 이용하여 값을 쓰면, 실제 Bar LED 하드웨어처럼, value에 따라 LED 불이 켜진다. Bar LED를 구성하는 8개의 LED는 value의 해당 비트가

1인 경우에 켜지도록 되어있다. 즉, BARLED1의 제일 아래 두 개의 불을 켜고 싶다면,

writeb_virtual_io(BARLED1, 0x03); 과 같이 하면 된다.


```
#define  BARLED1                0x408    // Output
#define  BARLED2                0x40C    // Output
```

3. 동작실험

실험에 앞서 실험 장치를 사용하기 위한 기본적인 코드는 핵심 부분은 다음과 같이 구성된다. 먼저 main() 함수는 Bar LED와 같은 가상 디바이스를 init_virtual_io()로 초기화하고, FreeRTOS의 태스크(쓰레드) 생성 함수인 xTaskCreate() API를 이용하여 실험 구동 함수 Exp_1_Task()를 FreeRTOS 태스크로 정의한다. init_virtual_io는 Bar LED를 포함하여 다양한 가상 다바이스를 초기화 하지만, 이번 실험에서는 Bar LED만을 사용한다.

마지막으로 vTaskStartScheduer()로 RTOS 스케쥴러를 구동한다. 이때부터 portTASK_FUNCTION()으로 정의된 태스크들이 실행된다.

```
int
main(void)
{
    init_virtual_io(ENABLE_LED);       // Enable Virtual IO Devices
    xTaskCreate(Exp_1_Task,
                        (const signed char * const)"Exp_1_Task",
                        2048,
                        (void *)NULL,
                        tskIDLE_PRIORITY + 1,
                        NULL);
    vTaskStartScheduler();          // Never returns
    while(1)
      ;
    return 0;
}
```

실험 내용은 두 개의 샘플 소스가 제공되는 함수, Exp_1_Sample_A(), Exp_1_Sample_B()에 이어, 과제인 Exp_1_Homework_A(), Exp_1_Homework_B()를 호출한 뒤, 모드 실험 항목이 끝나면 가상 디바이스를 끝내고, 무한 루프로 대기한다.

```
static
portTASK_FUNCTION(Exp_1_Task, pvParameters )
{
       while (1) {
            Exp_1_Sample_A();
            Exp_1_Sample_B();

            Exp_1_Homework_A();
            Exp_1_Homework_B();
       }
}
```

3.1 샘플 실험 A (Exp_1_Sample_A)

```
void
Exp_1_Sample_A(void)
{
    u16 sw;

    while (1) {
      sw = NDS_SWITCH();
      writeb_virtual_io(BARLED1, sw >> 8);
      writeb_virtual_io(BARLED2, sw & 0xFF);

      if (NDS_SWITCH() & KEY_START)
          break;
      vTaskDelay(50);
    }
    while (NDS_SWITCH() & KEY_START)
      vTaskDelay(10);        // Wait while START KEY is being pressed
}
```

Sample 실험 A는 닌텐도의 스위치 포트를 읽어 스위치의 상태를 그대로 Bar LED 두 개에 표시하는 프로그램으로 닌텐도의 스위치를 누르면, 누르고 있는 동안, 12개의 스위치에 해당하는 LED가 켜지는 것을 확인할 수 있는 프로그램이다. 소스에 보면, NSD_SWITCH() 값을 그대로 BARLED1, BARLED2에 출력하는 것을 확인할 수 있다. 이 샘플 프로그램의 소스는 ./source/Exp_Sample.c에 있다.

while(1) 문 안쪽의 KEY_START를 NSD_SWITCH()와 &하는 if 문은 START 버튼이 눌리면 while 문에서 빠져나가 결국 이 함수에서 return 하도록 한다. (첫번째 while 문에서 break로 빠져나오면, 다시 while 문에서 같은 KEY_START 스위치가 눌려 있는 동안 기다렸다가 스위치에서 손을 떼는 순간, 함수를 끝낸다) - 이 부분은 매번 실험이 여러 개의 단위 실험으로 이루어지기 때문에, 그 단계를 넘어갈 때, START 버튼을 눌러 구분하기 위한 것이다.

양쪽 while 문 모두에 vTaskDelay() 함수를 쓰는 것은 FreeRTOS에게 이 태스크보다 우선 순위가 낮은 다른 Task를 실행할 기회를 주는 것으로 이 함수를 호출하지 않으면, 이 태스크보다 우선 순위가 낮은 태스크들이 전혀 실행되지 않는다. 이 태스크보다 우선 순위가 낮은 태스크가 없은 CPU가 idle 상태에 놓이며, 대개는 전기 소모를 최소화하면서 대기한다.

3.2 샘플 실험 B (Exp_1_Sample_B)

```
void
Exp_1_Sample_B(void)
{
 u16 sw;
 u8  led_state = FALSE;
 u8  key_pressed = FALSE;

 writeb_virtual_io(BARLED1, 0);
 writeb_virtual_io(BARLED2, 0);
 while (1) {
      sw = NDS_SWITCH();

      if (((key_pressed == FALSE) && (sw & KEY_R))) {
           key_pressed = TRUE;
           led_state = !led_state;
           writeb_virtual_io(BARLED2, led_state);
      }

      if ((key_pressed == TRUE) && (!(sw & KEY_R)))
           key_pressed = FALSE;

      if (NDS_SWITCH() & KEY_START)
           break;
      vTaskDelay(50);
 }
 while (NDS_SWITCH() & KEY_START)
      vTaskDelay(10);        // Wait while START KEY is being pressed
}
```

샘플 실험 B는 뒤편 오른쪽 스위치(R)를 매번 누를 때 마다, BARLED2의 가장 오른쪽 LED가 켜졌다, 꺼졌다를 반복하는 프로그램이다. 즉, 누르고 있는 동안 켜지는 것이 아니라, 한번 누를 때마다 on, off를 반복하는 토글(toggle) 동작을 한다. 소스에서는 LED의 현재 상태와 스위치가 현재 눌려있는 상태라는 것을 유지하기 위한 상태 (key_pressed) 두 가지 변수가 추가로 사용된 것을 확인할 수 있다.

3.3 과제 A, B (Exp_1_Homework_A, Exp_1_Homework_B)

과제는 ./source/Exp_Homework.c의 Exp_1_Homework_A(), Exp_1_Homework_B() 함수를 지정된 내용의 동작을 하도록 작성하는 것이다.

과제 A : 프로그램이 시작되면, BARLED1의 LED 하나가 켜지고, KEY_LEFT, KEY_RIGHT를 누를 때 마다, 각각 왼쪽, 오른쪽으로 한 칸씩 켜진 LED가 옮겨 가는 것이다. 켜진 LED가 BARLED1의 오른쪽 끝이면 KEY_RIGHT를 눌러도 변화가 없어야 하며, 반대로 BARLED1의 왼쪽 끝에서는 KEY_LEFT를 눌러도 변화가 없어야 한다. 이 실험 동안 BARLED2는 모두 꺼진 채로 둔다.

과제 B : 위 과제 A를 BARLED1, BARLED2 모두를 이용해서 하는데, 이번엔 KEY_L, KEY_R을 이용한다. 또 BARLED1의 왼쪽 끝에서 KEY_L을 누르면 BARLED2의 오른쪽 끝 LED로 켜진 LED가 옮겨가고, 반대로 BARLED2의 오른쪽 끝에서 KEY_R을 누르면 BARLED1의 왼쪽 끝 LED가 켜지도록 한다. 또 BARLED1의 오른쪽 끝과 BARLED2의 왼쪽 끝에서 각각 KEY_L, KEY_R이 눌리면, 다른 Bar LED로 켜진 LED가 건너가도록 하여, 결국 LED가 KEY_ 좌우로 회전도록 동작시킨다.

(주의 사항)

모든 실험 중 언제라도 START 버튼을 누르면 다음 실험 단계로 가도록 (샘플 소스의 NDS_SWTICH() & KEY_START 비교 방법 사용) 프로그램 하라.

(제출 기한)

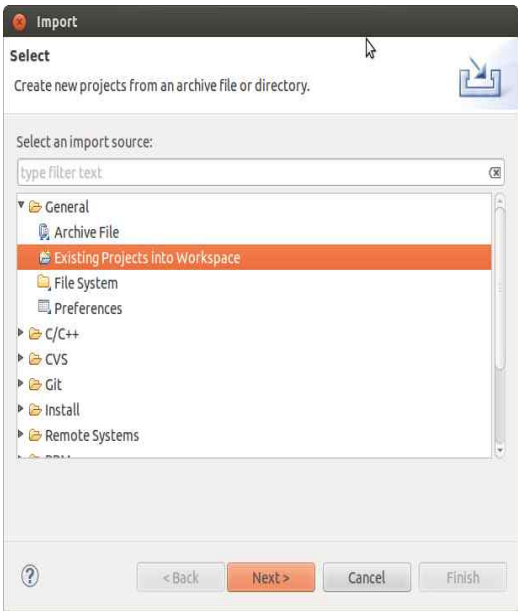
과제 A : 실험 당일

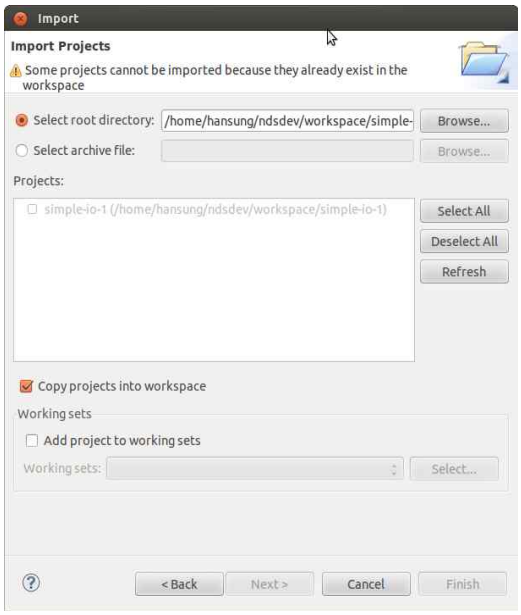
과제 B : 다음 실험 시간

4-1. 소스 및 샘플 프로그램 (강의 게시판 이용)

소스는 강의 자료실의 Simple-IO-1.zip를 이용하며, source 폴더의 Exp_Homework.c파일을 수정하여 과제를 수행한다.

소스는 workspace 폴더에 복사한 뒤, Eclipse의 [File] -> [import] 다이얼로그 박스에서 [Existing Projects into Workspace]를 선택하고 [Next->]를 누른뒤 (왼쪽 그림) [Select root directory]를 선택하고 [Browse...]에서 /home/hansung/ndsdev/workspace (소스 zip 파일을 설치한 곳)를 지정하고 프로젝트 선택하고 [Finish]하여 적용





B. 미리 만들어 놓은 Binary

이 실험 결과가 수행되는 모습은 Simple-IO-1.nds를 닌텐도 DS에 다운로드하여 실행하여 확인할 수 있음

4-2. 소스 및 샘플 프로그램 (Github 이용)

소스는 Github의 저장소에서 simple-io-1 소스를 check-out하여 과제를 수행한다.

https://github.com/hl1itj/nds-ide/tree/microprocessor/lab/simple-io-1

에서, Checkout하여 과제를 수행하고

https://github.com/hl1itj/nds-ide/tree/microprocessor/학번/simple-io-1

에서 각자 개발을 진행한다.